iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator

Alberto Núñez · Jose L. Vázquez-Poletti · Agustin C. Caminero · Gabriel G. Castañé · Jesus Carretero · Ignacio M. Llorente

Abstract Simulation techniques have become a powerful tool for deciding the best starting conditions on *pay-as-you-go* scenarios. This is the case of public cloud infrastructures, where a given number and type of virtual machines (in short VMs) are instantiated during a specified time, being this reflected in the final budget. With this in mind, this paper introduces and validates iCanCloud, a novel simulator of cloud infrastruc-tures with remarkable features such as flexibility, scalability, performance and usability. Further-more, the iCanCloud simulator has been built on the following design principles: (1) it's targeted to conduct large experiments, as opposed to oth-ers simulators from literature; (2) it provides a flexible and fully customizable global hypervisor for integrating any cloud brokering policy; (3) it reproduces the instance types provided by a given cloud infrastructure; and finally, (4) it contains a user-friendly GUI for configuring and launching simulations, that goes from a single VM to large cloud computing systems composed of thousands of machines.

Keywords Cloud computing · Cloud computing simulator · Cloud hypervisor · Validation · Scalability

A. Núñez (✉)
Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Madrid, Spain
e-mail: alberto.nunez@pdi.ucm.es

J. L. Vázquez-Poletti · I. M. Llorente Dep. de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain

J. L. Vázquez-Poletti
e-mail: jlvazquez@fdi.ucm.es

I. M. Llorente
e-mail: llorente@dacya.ucm.es

A. C. Caminero
Dep. de Sistemas de Comunicación y Control, Universidad Nacional de Educación a Distancia, Madrid, Spain
e-mail: accaminero@scc.uned.es

G. G. Castañé · J. Carretero Dep. de Informática, Universidad Carlos III de Madrid, Madrid, Spain

G. G. Castañé
e-mail: ggonzalez@inf.uc3m.es

J. Carretero
e-mail: jcarrete@inf.uc3m.es

1 Introduction

Cloud computing is a paradigm which provides access to a flexible and on-demand computing infrastructure, by allowing the user to start a required number of virtual machines (in short VMs) to solve a given computational problem. If the same software and configurations are needed, the VMs may be started using the same image. This way, a machine offered by a computing cloud may become whatever the user needs, from a standalone computer to a cluster or Grid node.

Nowadays, cloud computing systems are increasing their role due to the fast (r)evolution of computer networks and communication technologies. A very clear proof of this fact is that very important companies like Amazon, Google, Dell, IBM, and Microsoft are investing billions of dollars in order to provide their own cloud solutions [28].

As soon as the scientific community had access to cloud production infrastructures, the first applications started to run on the cloud [26, 34]. In many Research areas, the leap from traditional cluster and Grid computing to this new paradigm has been mandatory, being the main reason an evolution in the computational needs of the applications [10]. A remarkable fact from this evolution is that in a pre-cloud environment, hardware defines the level of parallelism of an application. In cloud computing, the level of parallelism is defined by the application itself, as there is no restriction in the number of machines, and CPU availability is 100% guaranteed by standard.

There are two main cloud infrastructure types. On the one hand, private clouds where the user's institution maintains the physical infrastructure. These cloud infrastructures can be built using virtualization technologies like Nimbus [11], OpenNebula [20], or Eucalyptus [22]. On the other hand, public clouds where the cloud service can be outsourced by paying in a deployed VM per unit of time basis. Some examples of these public clouds are ElasticHosts[1] and Amazon's Elastic Compute Cloud.[2]

In order to develop new proposals aimed at different topics related to clouds (for example, datacenter management [2], or provision of resources [15]), a lot of work and money is required to set up an adequately-sized testbed including different data centers from different organizations and public cloud providers. Even if automated tools exist to do this work, it would still be very difficult to produce performance evaluation in a *repeatable* and *controlled* manner, due to the inherent variability of the cloud. Therefore, it is easier to use simulation as a mean of studying complex scenarios.

This paper presents iCanCloud, a simulation platform aimed to model and simulate cloud computing systems. This simulator has several design principles: (1) iCanCloud can conduct large experiments, as opposed to other simulators from literature; (2) iCanCloud provides a flexible and fully customizable global hypervisor, which let users implement any brokering policies; (3) instance types provided by Amazon are included in the simulation framework.

The main contributions of this paper are: (1) the development of iCanCloud, a simulator for cloud computing environments, which is specially focused on the simulation of Amazon instance types; (2) the design and implementation of a flexible hypervisor module that provides an easy method for integrating both existent and new cloud brokering policies; (3) the validation of the simulator, which has been carried out by conducting a set of experiments on the simulator and comparing them with actual results using all the instance types provided by Amazon; (4) the comparison of iCanCloud with a mature cloud simulator (this being CloudSim [7]), which highlights iCanCloud's scalability and capacity to simulate large size cloud environments.

2 Cloud Simulation Tools in Computer Science

As we mentioned previously, simulations are essential for carrying out research experiments in cloud systems. Simulations have been widely used in different fields of computer science over the years. For instance, in the networking research area we can find NS-2 [31], DaSSF [18], OMNET++ [32], and OPNET [23], among others. These simulators are focused on network details, such as network protocols, path discovery, latencies, or IP fragmentation, but lack the details

[1] http://www.elastichosts.com/

[2] http://aws.amazon.com/ec2/

to simulate virtualization-enabled computing resources and applications.

The work [21] describes SIMCAN, a simulation platform for modeling HPC architectures. This platform is aimed to test both existent and new designs of HPC architectures and applications. SIMCAN has a modular design that eases the integration of the different systems on a single architecture. The design follows a hierarchical schema that includes simple modules, basic systems (computing, memory managing, I/O and networking), physical components (nodes, switches, etc) and aggregations of components.

For Grids, another set of simulators have been developed, such as GridSim [3], OptorSim [1], SimGrid [13] and MicroGrid [19], among others. These tools can simulate brokerage of resources, or execution of different types of applications on different types of computing resources, but as before they lack the details to simulate a cloud environment.

To the author's knowledge, the only tools that can simulate a real cloud system are CloudSim [7], MDCSim [17], and GreenCloud [16]. In the case of CloudSim [4], several research articles have been published with results obtained by using it [2, 5, 6, 15]. This tool was initially based on a Grid simulator [4] (this being GridSim [27]). So, a new layer on top of GridSim was implemented to add the ability to simulate clouds. For later versions, in-depth re-implementations took place including a full implementation of its simulation kernel so it does not rely on GridSim anymore.

But CloudSim still has some drawbacks. One of the main drawbacks of CloudSim is related to the fact that it is written in Java, because Java can only handle at most 2 GB of memory in 32 bits systems. This fact heavily affects the design of experiments, which must be large enough to extract interesting conclusions, but on the other hand must be small enough to fit into 2 GB of memory. However, this limitation does not affect 64 bits systems. Conversely, one of the design principles of iCanCloud is the ability to conduct large experiments, and this is why iCanCloud is written in C++. Thanks to this, iCanCloud can use all the memory available on the machines running the experiments, for 32 and 64 bits machines.

MDCSim [17] is an event driven simulation—similarly to CloudSim. The main drawback of MDCSim is that it is not available for public download since it is built on CSIM [9], a commercial product.

Regarding GreenCloud [16], this is an extension to the NS2 [31] network simulator. GreenCloud is focused on simulating the communications between processes running in a cloud at packet level. In the same way as NS2, GreenCloud is written in C++ and OTcl, and this is a disadvantage of this tool—two different languages must be used to implement one single experiment. GreenCloud provides plugins that allow the use of physical layer traces which make experiments more real, for instance, a packet loss probability in the optical fiber depending on the transmission range, which can be obtained via simulation of signal propagation dynamics and inserted into GreenCloud. iCanCloud in turn provides in-depth simulation of physical layer entities such as cache, allocation policies for memory and file system models, which help iCanCloud to create truly accurate experiments, although no power consumption model is in place at the moment.

One of the key points of cloud computing is the power consumption of datacenters. Each simulation tool provides a different support for this. In CloudSim, no power consumption model is implemented, but they provide the necessary functions to allow users implement their own power consumption models [7]. MDCSim has a power consumption model based on [35]. In this model, for a fixed operating frequency, the power consumption of the server is approximated using linear functions of the server utilization, which is calculated taking into account number of nodes, number of requests and average execution time of requests [17]. GreenCloud is specially designed to simulate power consumptions of the datacenter components (server, switches and links), so that efficient power consumption strategies can be developed. The power consumption model is based on [8], and takes into account all the different components of a datacenter, such as chassis or line cards of a switch, or frequency of a CPU. Regarding iCanCloud, by now it does not provide

models for power consumption, although this is included as future work.

None of the existing cloud simulator tools (CloudSim, MDCSim and GreenCloud) have a full GUI. CloudSim has CloudAnalyst [36], which allows the configuration of high level parameters. GreenCloud can be set to create trace files to be read using the network animation Nam [30], but no configuration can be done using it. As opposed to them, iCanCloud has a user friendly GUI which allows users create experiments easily.

Probably the main drawback of CloudSim, MDCSim and GreenCloud, which is other of the design principles of iCanCloud, is the fact that experiments on those simulators can only use the resources of one machine. For example, one experiment can only use the processors of one machine. So, if a cluster of machines is available to conduct experiments, each experiment must be executed in one machine. As opposed to them, iCanCloud has been designed to perform parallel simulations, so one experiment can be executed spanning several machines. Thanks to this, if a cluster of machines is available to conduct experiments, and in the case that an extremely large experiment is needed, this experiment can use all the machines in the cluster. The only requirement to conduct experiments spanning more than one machine is that machines must have MPI [29] installed. However, this feature is currently *Work-in-Progress* and it is not available yet. Table 1 depicts a summary of the cloud simulation tools reviewed above.

3 The iCanCloud simulation platform

The ever-increasing complexity of computing systems has made simulators a very important choice for designing and analyzing large and complex architectures. In the field of cloud computing, simulators become especially useful for calculating the trade-offs between cost and performance in *pay-as-you-go* environments. Hence, this work describes a flexible and scalable simulation platform for modeling and simulating large environments that represent, both actual and non-existent cloud computing architectures.

iCanCloud is a simulation platform aimed to model and simulate cloud computing systems, which is targeted to those users who deal closely with those kinds of systems. The main objective of iCanCloud is to predict the trade-offs between cost and performance of a given set of applications executed in a specific hardware, and then provide to users useful information about such costs. However, iCanCloud can be used by a wide range of users, from basic active users to developers of large distributed applications.

Although each user is intersected on different features provided by the cloud, all of them have the same objective: optimizing the trade-off between cost and performance, which is the real hard task iCanCloud tries to alleviate. Thus, this simulation platform provides a scalable, flexible, fast and easy-to-use tool which let users obtain results quickly in order to help to take a decision for paying a corresponding budget of machines.

Table 1 Summary of cloud simulators

Parameter	CloudSim	MDCSim	GreenCloud	iCanCloud
Platform	–	CSIM	NS2	OMNET, MPI
Language	Java	C++/Java	C++/OTcl	C++
Availability	Open source	Commercial	Open source	Open source
Graphical support	Limited (through CloudAnalyst [36])	None	Limited (through Nam [30])	Full
Communication models	Limited	Limited	Full	Full
Physical models	None	None	Available using plugin	Full
Models for public cloud providers	None	None	None	Amazon
Support for parallel experiments	No	No	No	WiP
Support for power consumption modeling	Limited	Yes	Yes	WiP

The proposed features are desirable for any simulation platform, but their meaning can be blurry depending on the context in which they are used. Scalability means whether the corresponding simulator is able to simulate large-scale systems without loosing performance. Likewise, performance determines the speed which a simulator executes a corresponding simulation. In general, the larger the size of the architecture to be simulated, the greater the time needed to execute the simulation. Moreover, a flexible simulator must let users build environments easily, using several component models with different levels of detail. In fact, the proposed hypervisor model let users to integrate any cloud brokering policy to manage a set of fully customizable VMs. Thus, different brokering policies can be fully customized by using this simulation platform.

The simulated cloud computing scenarios are modeled using a set of existent components provided by iCanCloud; they represent the behavior of real components that belong to real architectures like disks, networks, memories, file systems, etc. Those components are hierarchically organized within the repository of iCanCloud, which compose the core simulation engine. Besides designing simulated environments using components provided by iCanCloud, new components can be added to its repository. Moreover, iCanCloud allows an easy substitution of components for a particular component. Those interchangeable components can differ in level of detail (to make performance versus accuracy trade-offs), in the functional behavior of the component, or both.

3.1 Features

The most remarkable features of the iCanCloud simulation platform include the following:

1. Both existing and non-existing cloud computing architectures can be modeled and simulated.
2. A flexible cloud hypervisor module provides an easy method for integrating and testing both new and existent cloud brokering policies.
3. Customizable VMs can be used to quickly simulate uni-core/multi-core systems.
4. iCanCloud provides a wide range of configurations for storage systems, which include models for local storage systems, remote storage systems, like NFS, and parallel storage systems, like parallel file systems and RAID systems.
5. iCanCloud provides a user-friendly GUI to ease the generation and customization of large distributed models. This GUI is especially useful for: managing a repository of pre-configured VMs, managing a repository of pre-configured Cloud systems, managing a repository of pre-configured experiments, launching experiments from the GUI, and generating graphical reports. However, experiments can be executed also by using traditional command line scripts.
6. iCanCloud provides a POSIX-based API and an adapted MPI library for modelling and simulating applications. Also, several methods for modelling applications can be used in iCanCloud: using traces of real applications; using a state graph; and programming new applications directly in the simulation platform.
7. New components can be added to the repository of iCanCloud to increase the functionality of the simulation platform.

3.2 iCanCloud Design

The basic idea of a cloud computing system is to provide users a pseudo-customizable hardware environment where they can execute specific software. Therefore, in order to model entire cloud computing systems, the architecture of iCanCloud has been designed based on this principle. Thus, Fig. 1 shows the layered architecture of iCanCloud.

The bottom of the architecture consists of the hardware models layer. This layer basically contains the models that are in charge of modeling the hardware parts of a system, like disk drives, memory modules and CPU processors. Using those models, entire distributed systems can be modeled and simulated. In turn, this section consists of four groups, where each corresponds to a specific

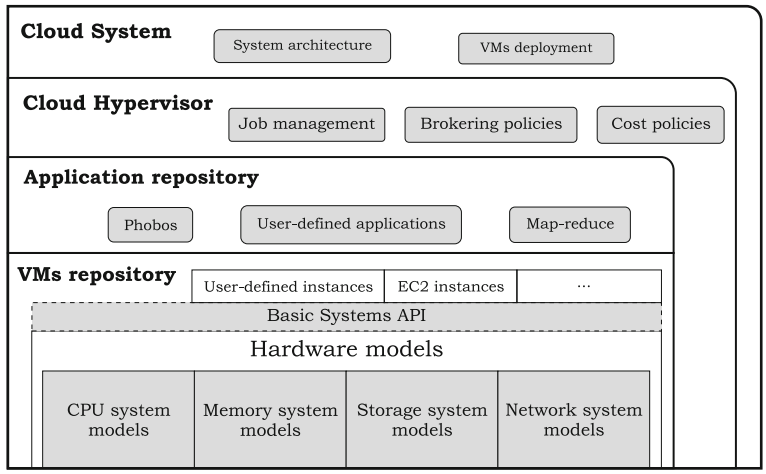


Fig. 1 Basic layered schema of iCanCloud architecture

basic system: processing system (CPU), memory system, storage system, and network system.

The basic system's API module is directly connected with the hardware models layer. Basically this module contains a set of system calls which are offered as an API (Application Programming Interface) for all applications executed in a VM modeled using iCanCloud. Thus, those system calls provide the interface between applications and the services provided by the hardware models. Moreover, researchers can write applications to be simulated in iCanCloud using this API. In order to maintain a certain degree of compatibility, this API pretends to be a subset of POSIX.

Following, the set of functions provided by this API is described for each basic system (see Listing 1).

Line 2 of Listing 1 shows a function for using the CPU service. Basically applications which request CPU processing must invoke the *iCanCloud_cpu* function to specify the corresponding amount of instructions to be executed, measured in MIs (Million Instructions). The CPU can contain one or more CPU cores.

The main task of the memory system is to assign the corresponding amount of memory to each application that requires it. Thus, this system receives requests for memory allocation and calculates where and how this memory has to be assigned. This feature is very useful for analyzing the amount of memory used for each application, especially in large distributed environments.

In order to interact with the memory system, applications must use the functions specified in lines 5 and 6 of Listing 1. Basically this interface consists of two functions. First function, called *iCanCloud_allocMemory*, is in charge of

```
1  // Functions provided by the computing system
2  void iCanCloud_cpu (long int numInstructions);
3
4  // Functions provided by the memory system
5  void iCanCloud_allocMemory (int memorySize);
6  void iCanCloud_freeMemory (int memorySize);
7
8  // Functions provided by the storage system
9  void iCanCloud_open (char* fileName);
10 void iCanCloud_close (char* fileName);
11 void iCanCloud_create (char* fileName);
12 void iCanCloud_delete (char* fileName);
13 void* iCanCloud_read (char* fileName, unsigned int offset, unsigned int size);
14 void* iCanCloud_write (char* fileName, unsigned int offset, unsigned int size);
15
16 // Functions provided by the network system
17 void iCanCloud_createListenConnection (int localPort, string type);
18 void iCanCloud_createConnection (string destAddress, int destPort, int id, string type);
19 void iCanCloud_sendDataToNetwork (iCanCloud_Message *sm, int id);
20 void iCanCloud_receiveDataFromNetwork (iCanCloud_Message *sm, int id);
```

Listing 1 Functions provided by the basic system's API

allocating memory. Second function, called *iCanCloud_freeMemory*, is in charge of freeing the previously allocated memory. Parameter *memorySize* indicates the amount of memory required (measured in bytes) to perform the corresponding operation.

The storage system is in charge of managing all accesses to data. The set of functions shown in lines 9–14 of Listing 1 offers an interface to interact with the storage system, which basically consists of a set of functions for managing files.

Functions *iCanCloud_open* and *iCanCloud_close* are in charge of opening and closing respectively a file given its name.

Functions *iCanCloud_create* and *iCanCloud_delete* are in charge of creating and removing respectively a file given its name.

Finally, function *iCanCloud_read* is in charge of reading data in the file specified in the parameter called *fileName*. Similarly, function *iCanCloud_write* writes data in a given file. The amount of data to be read or written is specified in the parameter *size*. Finally, the parameter *offset* specifies the starting point in the file where requested data is processed.

The network system is in charge of managing connections with other applications located in remote nodes, and also processing both the received and sent packets. The network system's API is shown in lines 17–20 of Listing 1. Using this interface, applications can manage connections with remote nodes, and send and receive data through the network.

Function *iCanCloud_createListenConnection* creates an incoming connection from a remote location. Otherwise, function *iCanCloud_createConnection* establishes a connection with a remote application.

Function *iCanCloud_sendDataToNetwork* sends data through the network system to a remote location. Similarly, *iCanCloud_receiveDataFromNetwork* receives data from a remote location. First function is in charge of sending data from the application that invokes this function to a remote application specified in the message *sm*. Second function is in charge of receiving data from a remote application specified in the message *sm*.

Besides functions provided by this API, iCanCloud also provides a high level layer for developing distributed applications. This layer is placed in the application component and provides standard interfaces for executing distributed applications. Currently iCanCloud has implemented an interface for executing MPI applications.

Upper layer consists of a VMs repository. This repository contains a collection of VMs previously defined by the user. Initially, the iCanCloud simulator provides few models of existing VMs in well known clouds like Amazon (EC2). Moreover, users can add, edit or remove VMs from this repository. Each VM is modeled by configuring the corresponding underlying hardware models for each basic system.

In a cloud system, the VM is the most relevant component. Similarly in iCanCloud, a VM is a building block for creating cloud systems. The key of this simulation platform is modularity, which let nested complex modules using other modules previously defined. Thence, the basic idea of iCanCloud consists on using VMs modules for building entire cloud computing systems.

In those systems, VMs are in charge of hiding the hardware details, providing to users a logic view that corresponds with the user requirements. Thus, the VMs models defined in this layer use the previously defined hardware components defined in the bottom layer.

Otherwise, the application repository contains a collection of pre-defined applications customized by users. Similarly to the repository of VMs, initially this repository provides a set of pre-defined application models. Those models will be used in order to configure the corresponding jobs that will be executed in a specific instance of a VM in the system. Moreover, new application models can be easily added to the system, because iCanCloud provides an API in order to ease the development of new application models.

Upper layer, called cloud hypervisor, consists of a module in charge of managing all incoming jobs and the instances of VMs where those jobs are executed. Once a job finishes its execution, this module sets as idle the VMs where that job has been executed, and then re-assign the available resources in the system to execute

the remaining jobs. This module also contains cost policies in order to assign incoming jobs to a specific instance calculated by the corresponding heuristic.

Finally, at the top of the architecture is the cloud system module. This module contains a definition of the entire cloud system, which basically consists on the definition of the hypervisor, and the definition of each VM that composes the system.

However, the key innovation of this simulation framework lies in a modular and flexible design. Figure 2 shows the UML 2.3 class diagram of the iCanCloud simulation platform.

This model is split in two different parts. On the one hand, dark grey squares represent the hardware part of the cloud environment. On the other hand, the light grey squares represent those modules in charge of managing the cloud system. The VM class acts as a link among the physical resources of the cloud environment, such as nodes and networks, and the resources used by users, giving them an illusion of using directly the physical resources to execute the corresponding jobs. Thus, depending of the configuration of each VM, those are mapped to the physical nodes existent in the cloud system model. The main module of this section is the hypervisor, which is the center piece of the system.

3.3 The Hypervisor Module

The hypervisor module is the master key of the iCanCloud simulation core. Basically, this module is in charge of managing the VMs and executing the jobs defined by users. In order to accomplish this task, the hypervisor can be fully configured by integrating customized brokering policies.

Figure 3 shows the global architecture of a cloud computing system modeled using iCanCloud. This figure shows three main entities: users, hypervisor and data centers. The hypervisor module is in charge of achieving four main tasks: (1) managing the VMs of the entire cloud system, (2) managing the list of jobs submitted by users, (3) scheduling those jobs to be executed in the corresponding VM instances, and (4) defining cost policies for each VM instance type. Data centers represent a set of Virtual Machines, each one configured with a pre-defined features such as CPU, storage, memory, and network. Finally, users are represented as entities that submit a set of jobs to be executed on specific VM instances. Those submissions arrive directly

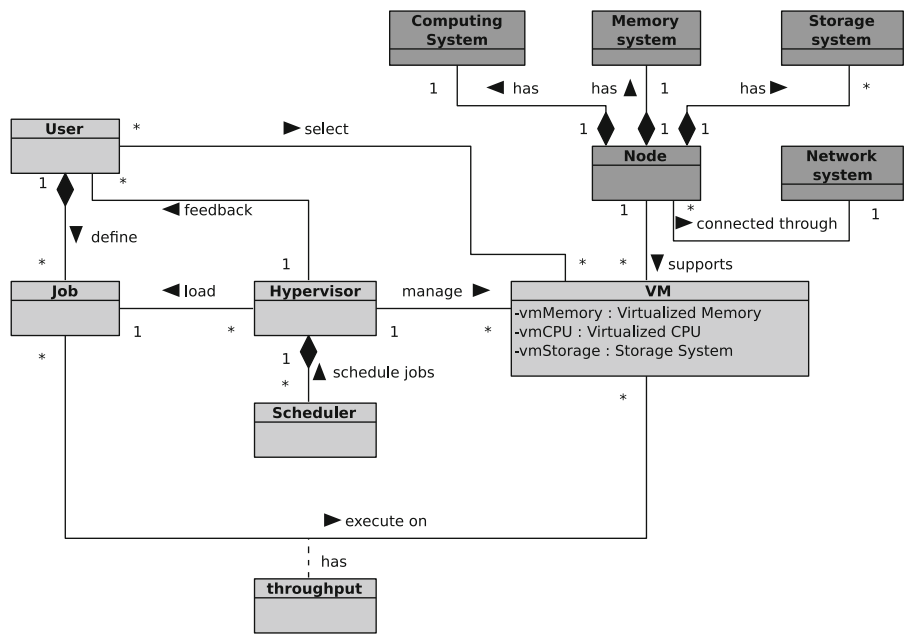


Fig. 2 UML class diagram of iCanCloud simulation platform

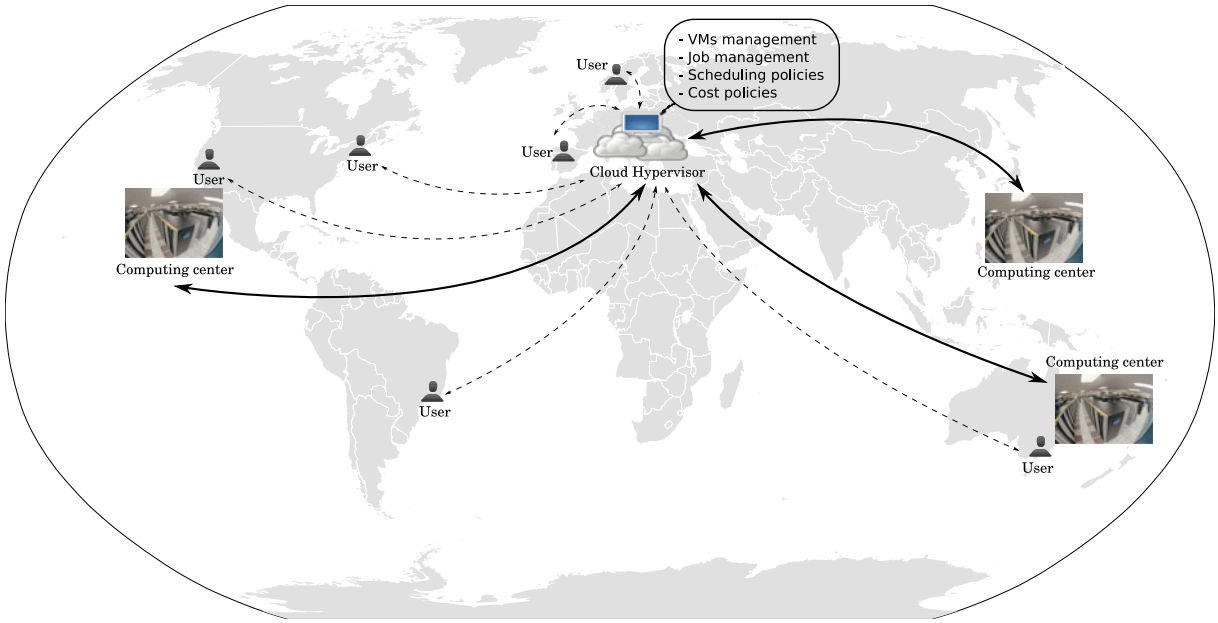


Fig. 3 Global schema of a scenario built using iCanCloud

to the hypervisor module. Then, the list of jobs is managed by this module to be executed in the specific VM instances hosted in the corresponding data center. Once the list of jobs submitted by a user is completely executed, a report is sent back to that user.

The main goal of the proposed hypervisor is two-fold: First, to provide a set of brokering policies to be customized easily depending of the system's requirements. Second, to allow the integration of custom brokering policies, such as cost-based policies, execution time-based policies and resource allocation-based policies.

In order to accomplish those goals, the proposed hypervisor has been designed to provide a high level of flexibility and usability. This module is implemented as an abstract class called *CloudHypervisor*. Also, this class provides a very

```
class CloudHypervisor:public iCanCloud_Base{

    protected:
        list <CloudJob*> waitingQueue;
        list <CloudJob*> finishQueue;
        list <CloudUser*> userList;
        list <VMInstance*> VMsMap;
        ...

    private:
        list <CloudJob*> get_job_list ();
        CloudJob* get_job (jobID);
        VMInstance get_vm (vmID);
        void insert_waiting_q (userID, jobID, vmID);
        void run_job (vmID, jobID);
        void move_qSrc_to_qDst (jobID, qSrc, qDst);
        ...

    public:
        void job_has_finished (jobID, results) = 0;
        jobID select_job () = 0;
        vmID select_vm () = 0;
        void job_has_been_inserted () = 0;
        void job_has_finished (jobID) = 0;
};
```

Listing 2 Interface provided by the CloudHypervisor class

intuitive interface to deal with the main tasks for managing a cloud system. This interface is shown in Listing 2.

Basically, this interface is divided in two different parts. First part consists of a set of methods implemented by *CloudHypervisor* (see lines 11–16 of Listing 2). Second part consists of a set of abstract methods that must be implemented by the corresponding subclass. Thus, in order to include a new brokering policy, users may just to create a class that inherits from *CloudHypervisor* and then implementing those methods shown in lines 20–24 of Listing 2. Of course, users may add new methods in the subclass.

The underlying idea of this subclass is to select the next job to be executed in the cloud, and the VM instances where that job will be executed. Thus, new scheduling policies can be easily integrated into the proposed hypervisor just by implementing the corresponding abstract methods. Hence, this subclass may also use methods provided by the hypervisor and the relevant information of each job submitted to the system. This information is basically the number of instructions to be executed, the size of the data to be read, and the output data to be written, which can be retrieved from each job object.

Main data structures are shown in lines 4–7 (see Listing 2). This class has two queues that contain the jobs submitted. First queue, called *waitingQueue* contains those jobs that have been submitted by users, but have not started its execution yet. Second queue, called *finishQueue*, contains those jobs that have finished their execution. The rest of queues must be declared in the subclass, which depend of the brokering policy implemented. For instance, using one queue for each VM instance type, or sharing a single queue for all jobs.

The method *get_job_list()* obtains a list of all submitted jobs; *get_job (jobID)* obtains the object associated with the identifier *jobID*; similarly, *get_vm (vmID)* returns the object corresponding to the VM *vmID*; *insert_waiting_q (userID, jobID, vmID)* inserts the job *jobID*, submitted by the user *userID*, to be executed in the VM *vmID* in *waitingQueue*; *run_job (vmID, jobID)* executes the job *jobID* in the VM *vmID*; finally, method *move_qSrc_to_qDst (jobID, qSrc, qDst)* moves the job *jobID* from the queue *qSrc* to queue *qDst*.

The rest of the methods must be implemented in the corresponding subclass, such as *job_has_finished (jobID, results)*, which indicates that the job *jobID* has finished with the feedback *results*; *select_job()* selects the next job to be executed; similarly, *select_vm()* selects the VM to execute next job; *job_has_been_inserted()* indicates that a new job has been submitted by a user; Finally, *job_has_finished(jobID)* indicates that job *jobID* has finished its execution.

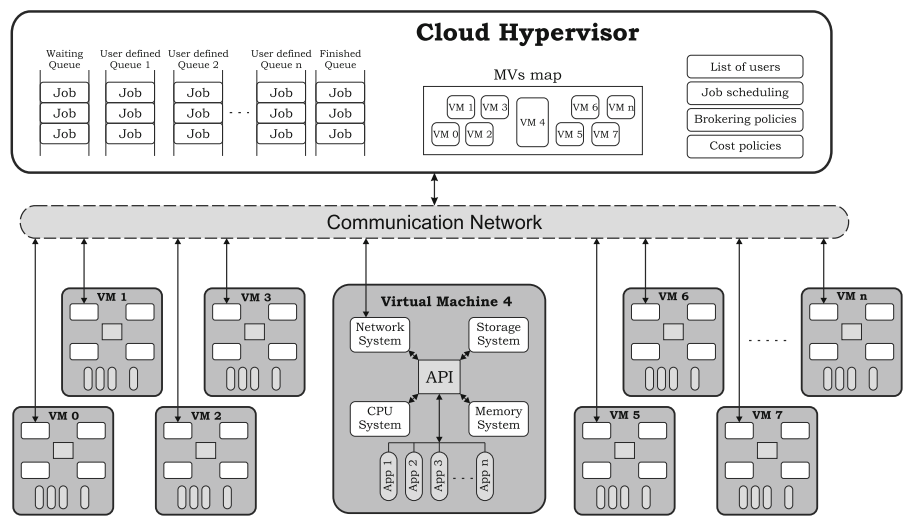


Fig. 4 Basic schema of the iCanCloud architecture

Figure 4 shows the basic schema of the proposed hypervisor module. Basically, the main parts of the hypervisor model are:

- Waiting queue: This queue handles jobs that have been submitted to the system, and are currently waiting to be executed.
- User-defined queues: Those queues contain those jobs that are currently being executed on any VM. Those queues must be declared on the subclass. Thus, depending of the brokering policy, the number of those queues may vary.
- Finished queue: This queue handles those jobs that have finished their execution.
- VMs map: This data structure contains relevant information of all those VMs that have been started up in the cloud system, such as the state of each VM, time-frame that each VM has been in an idle state, current job that is being executed in each VM, etc.
- List of users: List of users that have submitted jobs to the cloud system.
- Job scheduling and brokering policy: Policies required to manage jobs and VMs. Those policies must be implemented in the corresponding subclass of *CloudHypervisor*.
- Cost policy: Policy that establishes a price and budget to each VM instance type.

Figure 5 shows a sequence diagram of a typical use case, since a user sends a new job to the cloud system, until that job is done and the hypervisor collects the corresponding results.

3.4 iCanCloud Configuration

Creating simulated environments of distributed systems used to be a tedious and time-consuming

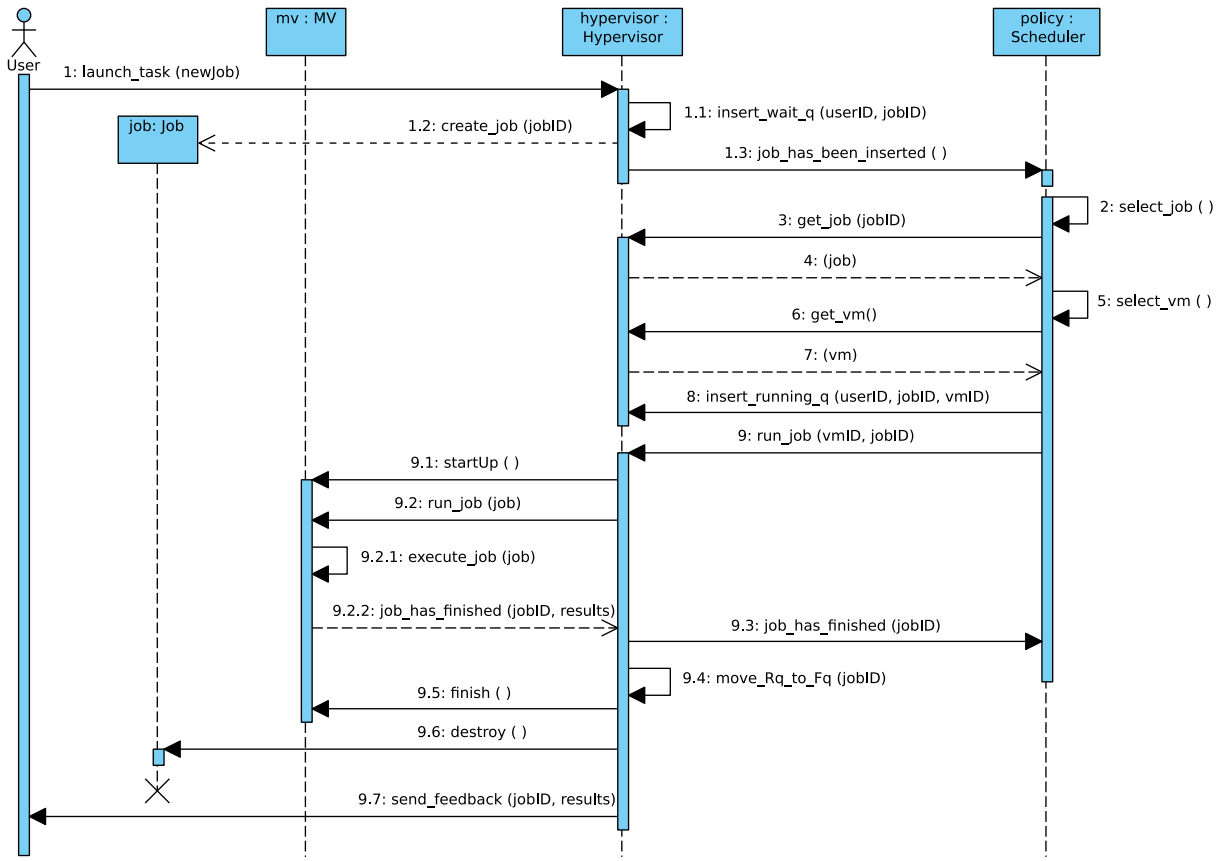


Fig. 5 Sequence diagram of the proposed hypervisor model

task. Basically this process consists on writing configuration files that contains both the definition of the components involved in the simulation, and a list of parameters that characterizes each one for a specific purpose. Moreover, the greater the size of the system to be modeled, the higher the complexity for modeling it. Thus, creating simulated environments requires both considerable amount of time and effort to be accomplished.

In this section, the process for creating simulated environments in iCanCloud is described in detail.

Simulated environments in iCanCloud are configured by using a *.ned* specification. Basically, this specification describes how many instances of each module have to be simulated. Moreover .ned specification is loaded at the beginning of the execution, whereof the simulation can be completely reconfigured without re-compiling the source code.

Figure 6 represents the steps since a cloud environment is modeled using the GUI provided by iCanCloud, until the simulation finishes its execution and the corresponding report is generated.

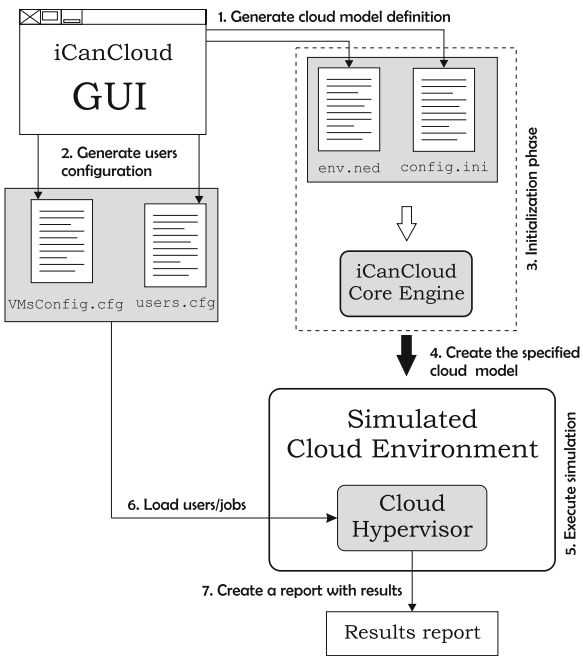


Fig. 6 iCanCloud configuration process

In iCanCloud, the definition and configuration of a modeled cloud system is divided in two different sections: cloud model definition and users configuration. First section defines the configuration of the cloud system. Otherwise, second section is in charge of defining the users in the cloud system, the jobs submitted by each user, and the VM instances requested to execute those jobs.

First section, denoted as cloud model definition, consists of two plain-text files: *env.ned* and *config.ini*. File *env.ned* defines the environment of the cloud model to be simulated. In other words, this file contains the name of the VM instances used in the cloud model, and the number of each instance to be simulated. This definition is specified using the NED language, which facilitates the modular definition of a distributed system. Moreover, NED files can be loaded dynamically into simulation programs, or translated into C++ by the NED compiler and linked into the simulation executable. Listing 3 shows a portion of a *env.ned* file where the Standard On-Demand Instances of Amazon EC2 cloud are modeled. This file shows a cloud model that contains three types of instances, and a different number of VMs for each. Specifically, this cloud model, called **cloudMode_AmazonEC2**, contains 10K small instances, 2K large instances, and 1K extra large instances.

Otherwise, file *config.ini* contains the configuration of each VM instance defined in the file *env.ned*, and the definition of the cloud hypervisor. Basically, the configuration of each instance consists on assigning values to the set of parameters of each basic system, such as CPU, memory and storage. Those parameters can take string,

```
1  import iCanCloud.VirtualMachines.VMs.*;
2  ...
3
4  network cloudModel_AmazonEC2{
5
6    submodules:
7
8      // Definition of 10K Small instances
9      VMs_small[10000]: VirtualMachine_Small;
10
11     // Definition of 2K Large instances
12     VMs_large[2000]: VirtualMachine_Large;
13
14     // Definition of 1K Extra-large instances
15     VMs_xlarge[1000]: VirtualMachine_ExtraLarge;
16     ...}
```

Listing 3 Definition of Amazon EC2 instances (env.ned)

numeric or boolean values, or can contain XML data trees. Numeric values include expressions using other parameters and calling C functions, random variables from different distributions, and values input interactively by the user. Numeric-valued parameters can be used to construct topologies in a flexible way. Listing 4 shows a portion of a *config.ini* file where the instances defined in Listing 3 are configured. The configuration of the hypervisor basically consists on specifying the hypervisor type used in this simulation. In this example, the *Min-Min_CloudHypervisor* is used (see line 6 of Listing 4).

Second section, denoted as users configuration, contains information about the users, the list of jobs submitted to the cloud system, and the set of VMs requested to execute the previously defined jobs. Basically, this information is structured in two different plain-text files: *users.cfg* and *VMsConfig.cfg*. Listings 5 and 6 show the syntax used to create those files.

File *users.cfg* contains the list of users that are currently submitting jobs in the modeled cloud system. Each user must define both a list of jobs to be executed in the cloud system, and the corresponding configuration of VMs where each

```
1  // Definition of user 1
2  User_1
3
4  // Definition of Job 1
5  JobType_1:[VMinstance|?]:[#VMs|?]
6
7  // Parameters for configuring Job 1
8  param_1:value_1
9  param_2:value_2
10 param_p:value_p
11
12 // Definition of Job 2
13 JobType_2:[VMinstance|?]:[#VMs|?]
14
15 // Parameters for configuring Job 2
16 param_1:value_1
17 param_2:value_2
18 param_p:value_p
19 ...
20
21 // Definition of Job 3
22 JobType_2:[VMinstance|?]:[#VMs|?]
23
24 // Parameters for configuring Job 3
25 param_1:value_1
26 param_2:value_2
27 param_p:value_p
28 ...
29
30 // Definition of user n
31 ...
```

Listing 5 Syntax of file users.cfg

```
1  [General]
2  network = cloudModel_AmazonEC2
3  ...
4  // --- Configuration of cloud hypervisor
5  cloudModel.hypervisor.type = "Min-Min_CloudHypervisor"
6
7  // --- Configuration of Small instances
8
9  // Configuration of CPU
10 cloudModel.VMs_small[*].numCPUs = 1                          // Number of CPU cores
11 cloudModel.VMs_small[*].cpuModule.CPUcore[*].speed = 9500   // In MIPs
12
13 // Configuration of storage
14 cloudModel.VMs_small[*].numDisks = 1                        // Number of disks
15 cloudModel.VMs_small[*].bsModule[*].diskType = "Disk_400GB" // Disk type
16
17 // Configuration of memory
18 cloudModel.VMs_small[*].osModule.memory.readLatencyTime_s = 0.000004    // Read latency (in sec.)
19 cloudModel.VMs_small[*].osModule.memory.writeLatencyTime_s = 0.000004   // Write latency (in sec.)
20 cloudModel.VMs_small[*].osModule.memory.size_MB = 1700                  // Memory size (in MB)
21
22 // --- Configuration of Large instances
23 cloudModel.VMs_large[*].numCPUs = 2                          // Number of CPU cores
24 cloudModel.VMs_large[*].cpuModule.CPUcore[*].speed = 19000  // In MIPs
25 cloudModel.VMs_large[*].numDisks = 1                        // Number of disks
26 cloudModel.VMs_large[*].bsModule[*].diskType = "Disk_400GB" // Disk type
27 cloudModel.VMs_large[*].osModule.memory.readLatencyTime_s = 0.000004    // Read latency (in sec.)
28 cloudModel.VMs_large[*].osModule.memory.writeLatencyTime_s = 0.000004   // Write latency (in sec.)
29 cloudModel.VMs_large[*].osModule.memory.size_MB = 7500                  // Memory size (in MB)
30
31 // --- Configuration of X-large instances
32 cloudModel.VMs_xlarge[*].numCPUs = 4                          // Number of CPU cores
33 cloudModel.VMs_xlarge[*].cpuModule.CPUcore[*].speed = 19000  // In MIPs
34 cloudModel.VMs_xlarge[*].numDisks = 1                        // Number of disks
35 cloudModel.VMs_xlarge[*].bsModule[*].diskType = "Disk_400GB" // Disk type
36 cloudModel.VMs_xlarge[*].osModule.memory.readLatencyTime_s = 0.000004   // Read latency (in sec.)
37 cloudModel.VMs_xlarge[*].osModule.memory.writeLatencyTime_s = 0.000004  // Write latency (in sec.)
38 cloudModel.VMs_xlarge[*].osModule.memory.size_MB = 15000                 // Memory size (in MB)
39 ...}
```

Listing 4 Configuration of Amazon EC2 instances (config.ini)

job must be executed. Thus, this file consists of different blocks, one for each user. Similarly, each block contains a set of jobs. Listing 5 shows the syntax used to create file *users.cfg*.

Each block in this file begins with the user ID. In this example, line 1 contains the definition of first user, denoted as *User_1*. Moreover, each block contains a set of jobs, whereof each job must contain two parts. First part consists of the name of the job, the type of VM instances where the job must be executed and the number of VMs used to execute the job (see line 5 of Listing 5). In this part, both instance types and the number of instances may be unset. Then the character '?' is used. In this case, the hypervisor is in charge of calculating the corresponding instance type or/and the number of them to execute the corresponding job.

In fact, in many cases users have no expertise to select both the instance type and the number of instances to execute a given job. In such situations, how many VMs of a particular instance are required to execute a job? 1000 small instances are enough? 100 large instances? Thus, depending of the brokerage policy used by the hypervisor, users have the option to delegate that responsibility to the hypervisor, which provides a high level of flexibility because different policies must be used to schedule jobs and provisioning the idle resources if the cloud.

Finally, the second part of the job definition consists of a list of pairs (parameter's name, parameter's value) corresponding to the set of parameters of current job. Thus, depending of the job type, different set of parameters must be configured.

File *VMsConfig.cfg* contains the instance types and number of VMs existent in the cloud model. This information is basically an overview of the information contained in the file env.ned. But, due to practical reasons, this file has been created instead of writing a parser to read the *env.ned* file and gather such information.

```
1 instance_1:number of VMs of type instance_1
2 instance_2:number of VMs of type instance_2
3 ...
4 instance_3:number of VMs of type instance_n
```

Listing 6 Syntax of file VMsConfig.cfg

Those configuration files are automatically generated by the iCanCloud GUI (to see some screenshots check the website http://www.icancloudsim.org/). Once the user has configured properly the simulated cloud environment, this GUI performs two steps. In the first step, the configuration files corresponding to the cloud model definition are generated: *env.ned* and *config.ini* (see step 1 of Fig. 6). Then, files corresponding to the user configuration section are generated: *VMsConfig.cfg* and *users.cfg* (see step 2).

Once those four configuration files are created, the iCanCloud simulation is launched, and then the initialization phase is performed (see step 3). In this phase, the iCanCloud core engine receives two files as input (*env.ned* and *config.ini*) which contain the definition of the experiment. The objective of this phase is to use that information to create the simulated environment previously defined in the GUI, which is performed by the core engine of iCanCloud (see step 4). Note that different cloud environments with different configurations can be launched without re-compiling the simulation platform neither changing the source code. Only the configuration files have to be modified, but this task is performed automatically by the GUI, which is totally transparent to the user. Those files are processed before executing the simulated environment.

Thus, the simulated cloud environment is generated, the simulation starts its execution (see step 5). In this step, all data structures are created and initialized in run-time. The most important module in this step is the cloud hypervisor. Basically, this module contains all the information required to manage completely the cloud system. This information is located in the files *VMsConfig.cfg* and *users.cfg*. Thus, when the cloud environment has been initialized, the hypervisor module reads those files to create and initialize the corresponding data structures described in Fig. 4 (see step 6).

Then, once the initialization phase has been fulfilled, the execution of the cloud model starts. This execution finalizes when all jobs defined in the file users.cfg have been executed completely. Then, a report containing the results of the simulation is generated (see step 7).

3.5 The iCanCloud Core Engine

The main principle of the iCanCloud simulation platform is to build a cloud computing system using Virtual Machines as building blocks, where each VM is defined by configuring the four basic systems: CPU, memory, storage and networking. Therefore, those basic systems are described in this section.

3.5.1 The CPU System

The computing system has been modeled in iCanCloud using 2 different components: the processor and the CPU scheduler. The strategy used for modeling this system is based on calculating the amount of time needed for executing the specific instructions invoked by applications. Therefore, CPUs are parameterized with a specific CPU processing power measured in MIPS (Million Instructions Per Second), which is a method of measuring the raw speed of a computer's processors. Similarly, the amount of computing invoked by applications is measured in MIs (Million Instructions).

The processor is the component in charge of calculating the amount of time spent in executing instructions invoked by applications. Each processor consists of a finite set of CPU cores, where the speed of all those cores is the same that the speed of its processor. Thus, several applications can be executed in parallel using the same processor, even when the number of applications executed at any given instant is greater than the number of CPU cores.

Otherwise, the CPU scheduler is the component in charge of managing all computing blocks to be executed in the corresponding CPU cores efficiently. Thus, this scheduler has been modeled using the time sharing technique, where several applications simulate their executions by multiplexing the CPU time because the computing blocks belonging to the simulated applications are divided in slices. Therefore, in single-core processors, only one application can be executed at any given instant.

Currently iCanCloud provides three different strategies for managing blocks in the simulated CPU scheduler: FIFO, Round-Robin, and priority-based policy. Moreover, new strategies can be added to the simulation platform.

3.5.2 The Memory System

The memory system has been modeled using two different components. On the one hand, a component called physic memory is in charge of simulating the physical characteristics of the memory. Those characteristics include mainly the size of the memory space and latency times. On the other hand, another component called memory manager is in charge of managing the memory accesses. Currently, this component manages the memory accesses needed for applications and disk cache.

Moreover, the memory system divides the memory space for two different purposes: memory used for application space and memory used for disk cache space. The main tasks performed by the memory manager are:

- Managing memory accesses from applications.
- Allocating memory pages required by applications.
- Freeing memory pages requested by applications.
- Managing a cache system for disk data.

3.5.3 The Storage System

The I/O system is usually a system bottleneck in most of the computing systems [24]. In iCanCloud, the storage system has been modeled using three modules:

- File system.
- Volume manager.
- Disk drives.

File system is in charge of translating data requests from applications to a list of blocks which contains the requested data.

Volume manager is a software component in charge of performing read and write operations of data blocks, sent from file systems. Basically, this component receives data block requests from file systems and it must redirect the incoming data block requests to the disk that contains such data. Depending of the modeled architecture, this

component can use the appropriate strategies for scheduling data block requests, like FIFO, elevator algorithm, and CSCAN.

Finally, disk drive is in charge of storing data blocks. Basically this component calculates the amount of time spent for processing the data block operations that came from volume manager.

Additionally, iCanCloud provides several configurations for using parallel I/O architectures such as RAID systems and parallel file systems.

3.5.4 The Network System

In iCanCloud, the network system let applications executed in Virtual machines interchange data through a communication network.

In order to fulfill this task, the INET framework has been used. This framework contains modules for simulating completely a network system, including network protocols like TPC or UPD. The main advantage of this method is the high level of accuracy obtained, because all elements that compose a network are simulated. However, the main drawback is performance, because this high level of detail needs a considerably CPU power to be calculated.

Moreover, the INET framework provides a set of modules such as routers, switches and network protocols, for building a wide range of networks. Thus users can build several network architecture models like LAN (Local Area Networks) and WAN (Wide Area Network). Moreover, those networks can be configured as wired and wireless networks, using Ethernet and Wi-Fi respectively.

3.6 Proof Case: Implementation of Provisioning and Scheduling Algorithms

To illustrate the usefulness of this work, two provisioning and scheduling algorithms (namely, Cloud Min-Min and Cloud Max-Min) are implemented using iCanCloud. The algorithms are divided in two main phases:

1. Provisioning, where a specific type and number of EC2 instances are deployed.
2. Scheduling, where tasks are matched with available resources.

In order to perform the provisioning, [33] i n - troduced a metric named Cost per Performance (C/P). This metric relates the needed takes to execute a number of tasks in Amazon EC2, with the cost required to hire the machines that pro-duce this execution time. This way, the best in-frastructure setup would be that which produced the lowest C/P value.

3.6.1 Cloud Min-Min

In the Cloud Min-Min algorithm, the n tasks with shortest execution time are taken for obtaining its best virtual infrastructure deployment. A sweeping is made with different values of n to achieve the best value for the C/P metric. In the case of having similar values, decision turns to less execution time or to less cost. Task scheduling follows the same procedure as the original Min-Min algorithm [12].

The pseudocode of this heuristic shown in Algorithms 1, 2 and 3. Initially, an array of execution times is created following Algorithm 1, which uses the job size in millions of instructions (MI) and VMs power in millions of instructions per second (MIPS).

Algorithm 1 Creation of arrays of execution times.

```
1: Let N = the number of tasks.
2: Let x = the number of instance types of VM.
3: Let T^v = the task execution time vector for
   the instance type v.
4: Let jobID = the identificator for a job.
5: Let v = the identificator for a VM.
6: Let vm = a VM.
7: Let job = a job.
8: jobList = get_job_list()
9: for jobID = 0 to jobList.size() − 1 do
10:    for v = 0 to x − 1 do
11:       job = get_job(jobID)
12:       vm = get_vm(v)
13:       T^v(jobID) = job.getMI()/vm.getMIPS()
14:    end for
15: end for
```

After that, Algorithm 2 provides the resource provisioning phase. As the resource provisioning consists in choosing the best configuration, a loop is made for using a different number of j tasks

Algorithm 2 Resource provisioning phase.

1: Let N = the number of tasks.
2: Let x = the number of instance types of VM.
3: Let T^v = the task execution time vector for the instance type v.
4: Let $V_{v,n}$ = the set containing n virtual machines of a v instance type.
5: Let j = the number of tasks used each loop in the resource provisioning phase.
6: Let CP = the vector that keeps the C/P metric for each j for all the instance types.
7: Let CP_{temp} = the temporary vector that keeps the C/P metric for each instance type for a given j.
8: Let P_v = the vector that keeps the execution time for the j tasks with minimum execution time for the instance type v.
9: Let *itype* = the instance type of VMs chosen.
10: Let n = the number of VMs chosen.
11: { *** Resource provisioning phase *** }
12: **for** $j = N$ downto 1 **do**
13: {Creation of temporal arrays of execution times}
14: **for** $v = 0$ to $x - 1$ **do**
15: $T^v_{\text{temp}} = T^v$
16: **end for**
17: {Creation of P_v arrays}
18: **for** $k = 0$ to j **do**
19: **for** $v = 0$ to $x - 1$ **do**
20: insert($P_v[k]$, min(T^v_{temp}))
21: remove(T^v_{temp}, min(T^v_{temp}))
22: **end for**
23: **end for**
24: {Calculate the temporal CP metric}
25: **for** $v = 0$ to $x - 1$ **do**
26: setN (j, v)
27: $CP_{\text{temp}}[v]$ = time($V_{v,n}$) × cost($V_{v,n}$)
28: **end for**
29: {Finally, calculate the CP metric}
30: **if** |min(CP_{temp})| > 1 **then**
31: $CP[j]$ = min(time($V_{v,n}$)) **or** min(cost($V_{v,n}$))
32: **else**
33: $CP[j]$ = min(CP_{temp})
34: **end if**
35: **end for**
36: createInstances($V_{itype,n}$) producing min(CP)

Algorithm 3 Task scheduling phase.

1: Let $minJobID$ = the identificator for the job with the lowest execution time.
2: Let job = the job with the lowest execution time.
3: Let $V_{v,n}$ = the set containing n virtual machines of a v instance type.
4: Let *itype* = the instance type of VMs chosen.
5: Let T^{itype} = the task execution time vector for the instance type *itype*.
6: Let n = the number of VMs chosen.
7: Let $vmID$ = the identificator of a VM.
8: {Initially, submit the first n jobs to the n VMs}
9: **for** $vmID = 0$ to n **do**
10: $minJobID$ = getMin(T^{itype})
11: $job = get\_job(minJobID)$
12: $run\_job(vmID, job)$
13: $vmID++$
14: **end for**
15: {If there are more jobs than VMs, submit jobs to VMs as VMs become idle}
16: **while** $getJobList().size > 0$ **do**
17: **if** $checkIdleVM(vmID) == true$ **then**
18: $minJobID$ = getMin(T^{itype})
19: $job = get\_job(minJobID)$
20: $run\_job(vmID, job)$
21: **end if**
22: $vmID = select\_vm()$
23: **end while**

from the total tasks N to be scheduled (line 12). Then, temporal arrays containing the execution times of tasks in each VM instance are created, to be used during the execution of the algorithm (line 14).

Next, another loop creates a vector (P_v) for each instance type v with the j shortest tasks (line 20) and removes the shortest task from the array of execution times of the given v (line 21). As can be seen in line 20, each P_v is ordered from the lowest execution time (held at $P_v[0]$) to the highest (held at $P_v[j-1]$). This line is one of the few differences between the Cloud Min-Min algorithm and Cloud Max-Min.

Then, a temporal CP metric is calculated. This loop is performed to set the number of instances for each instance type provided by

Amazon EC2 so that the j tasks are executed in parallel (line 26). This followed by the calculation of C/P metric, which is obtained for each instance type (line 27). Being the smallest value for this metric the best one, it may happen that it is achieved by several combinations of n instances of type x. At this point with equal C/P values, the setup is chosen considering the shortest execution time or the lowest cost (line 31). If there is only one configuration of instance types yielding the minimum C/P value, it is chosen as the C/P metric for this j (line 33).

At the end of the provisioning main loop, there will be a $CP[j]$ vector that contains the best C/P values from the previous iterations—this is, for each number of tasks j considered. To conclude this algorithm, the infrastructure with n machines of type x with the minimum C/P value is then instantiated (line 36).

Finally, the scheduling phase depicted by Algorithm 3 is performed. Initially, the jobs with the n lowest execution times are submitted to the n VMs (line 9). If there are more jobs than VMs, jobs are submitted to VMs as they become idle from the executions of previous jobs (line 16).

3.6.2 Cloud Max-Min

This algorithm is similar to Cloud Min-Min presented above, and is not presented in detail for space limitations. The only differences are (1) in Algorithm 2, line 20, in which the job with the lowest execution time is inserted at the end of P_v array; and (2) in Algorithm 3, lines 10 and 18, which choose the job with the highest execution time.

4 Validation of iCanCloud

After a simulator has been developed, implemented, and debugged, it must be tested for correctness and accuracy. However, determining that a simulator is absolutely valid over the complete domain of its whole intended field of applicability is a very hard and time-consuming task. Thus, the level of accuracy of a given simulator cannot be calculated for the entire domain this simulator is targeted using a single value, because this accuracy depends directly of the system to be modeled.

In the paper a validation process has been conducted to demonstrate the applicability and usefulness of the iCanCloud simulator. We focused on a critical application used by the Finnish-Russian-Spanish future Mission to Mars [14], which calculates the trajectories of Phobos [25], the Martian moon. Pertaining to the parameter sweep execution profile, the application divides the overall tracing interval in subintervals that are calculated by the subsequent tasks in the cloud – thus the tracing interval of a task is not related to its execution time. The tracing interval processed by each task is the same and the system performs dynamic scheduling where a continuous polling of free cores guarantees a constant resource use.

The present contribution culminates the evaluation process that started with the porting of the application to the Amazon EC2 public cloud infrastructure, formulation and validation of an execution model and a metric combining cost and performance [33]. This is accomplished by firstly comparing the results obtained from the mathematical model with those from iCanCloud. Finally, results from real executions in the Amazon EC2 infrastructure will be also used for the comparison.

The Phobos application has been chosen for the validation of iCanCloud because this application has undergone a research path as follows. First, this application has been ported to a cloud system. Next, a mathematical model of this application has been developed [33]. And finally, using the ported application and the mathematical model, a validation of the iCanCloud simulation has been performed. Thanks to using this application, a more detailed validation of the iCanCloud has been performed, since we compare results from iCanCloud with actual results from the ported application and results from the validated mathematical model.

As was explained before, the chosen cloud infrastructure is Amazon EC2, which has become the de facto standard public cloud infrastructure for many scientific applications. The baremetal infrastructure providing the services is located in two locations in USA, one in Asia and another one in Europe. The users may choose from a wide

Table 2 Characteristics of the different machine types offered by Amazon EC2. C.U. corresponds to EC2 Compute Units per core, the equivalent to a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor

Machine type	Cores	C.U.	Memory (GB)	Platform (bit)
Standard on-demand instances				
Small (Default)	1	1	1.7	32
Large	2	2	7.5	64
Extra large	4	2	15	64
High CPU on-demand instances				
Medium	2	2.5	1.7	32
Extra large	8	2.5	7	64

range of machine images that can be booted in one of the offered instance types. Depending the chosen instance type, the number of CPU core number, core speed, memory and architecture differ, as shown in Table 2. The speed per CPU core is measured in EC2 Compute Units, being each C.U. equivalent to a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. Nevertheless, accessing to an almost infinite computing infrastructure has its price, which depends on the instantiated VM type per hour.

The paper which describes the porting of the Phobos tracing application [33] introduced and validated an execution model, along with a study of the best infrastructure setup by means of instance types and number. In order to deal with the complexity level added by an infrastructure following a pay-as-you-go basis, the used C/P metric was defined as:

$$C/P = CT = \frac{C_h T_{exe} I}{i N_c^2} \left\lceil \frac{T_{exe} I}{i N_{vm} N_c} \right\rceil \quad (1)$$

where T_{exe} is the task execution time, the values of I and i correspond to the whole tracing interval and the tracing interval per task, that is, the grain of the application. On the other hand, N_{vm} and N_c are the number of Virtual Machines and number of cores per Virtual Machine, as shown in Table 2 along with the machine's usage price per hour (C_h). This way, the best infrastructure setup would be that which produced the lowest C/P value.

Figures 7, 8, 9, 10 and 11 present the results obtained by executing the model of Phobos application along with the results of the same application implemented on iCanCloud. Each figure represents the C/P metric for the experiments, where the small, large, extra large, high CPU medium and high CPU extra large instance types provided by Amazon are used, and number of VMs and tracing intervals are varied.

Mainly, the most relevant difference between the iCanCloud and the mathematical model is the variations obtained when the number of VMs increases. In those results obtained using iCancloud, we can see that in some cases, using the same size for the interval (in years) and increasing the number of VMs, causes an increase in the C/P metric, which does not happen in experiments using the mathematical model of Amazon. This is mainly caused because the cost of the each VM is measured in completed hours, whereof an hour

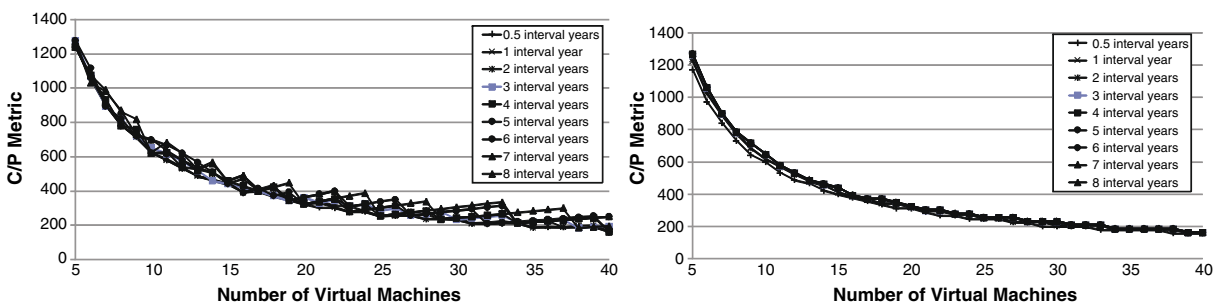
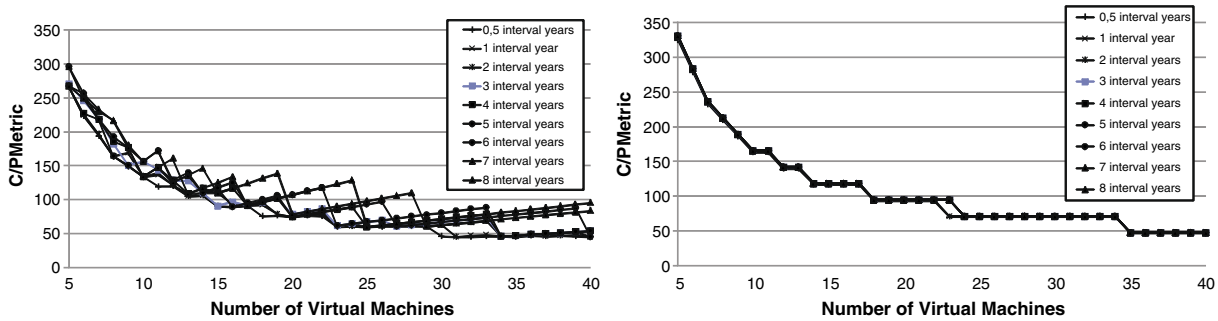


Fig. 7 Simulation versus mathematical model of Phobos using small instances



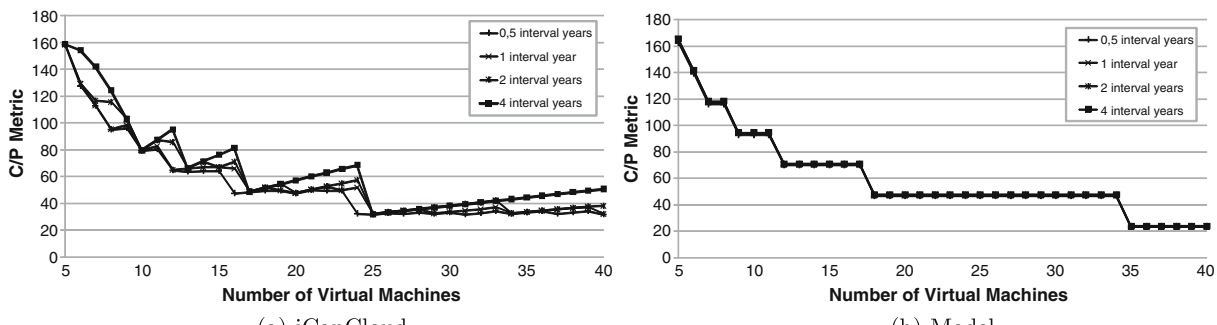
(a) iCanCloud (b) Model

Fig. 8 Simulation versus mathematical model of Phobos using large instances

cannot be split in fractions. Then, increasing the number of VMs provides the same execution time, which produces a increasing of the cost for this configuration. Logically, the greater number of VMs used, the greater cost of the system. This effect only appears when the number of VMs gets higher. When the number of VMs is low, the performance gain when more VMs are used justifies the increase in the cost. Moreover, the mathematical model does not represent the time spent on performing I/O operations.

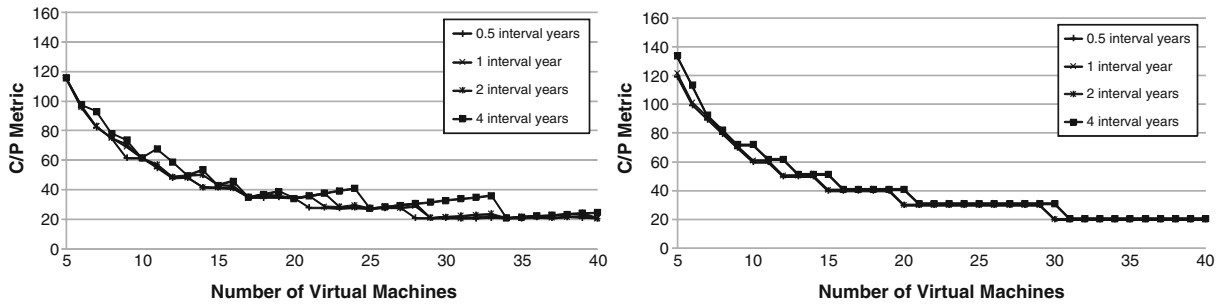
The best results are obtained in those tests that use small and high CPU medium instances of VMs (see Figs. 7 and 11). In those cases, the cost of increasing the number of VMs for reducing the execution time and then using less complete hours for the execution is similar to the cost of using less VMs that requires more complete hours to execute the testbed. It can be shown by comparing the simulations and the model, whereof both results show practically the same shape.

Otherwise, Figs. 8, 9 and 11 show a more noticeable difference between the simulation and the model. The difference that can be appreciated in those charts is because the C/P metric values are low, and then the differences between the simulation and the model are shown as peaks. Those peaks basically represent the maximum value of C/P, which means that the time required for executing this testbed consumes a little portion of the last hour of the total execution time. Thus, it means that using more VMs the testbed can be executed without requiring that portion of hour, and then there is no need of paying for an entire hour, obtaining a considerable drop in the C/P value.



(a) iCanCloud (b) Model

Fig. 9 Simulation versus mathematical model of Phobos using X-Large instances



(a) iCanCloud (b) Model

Fig. 10 Simulation versus mathematical model of Phobos using high CPU medium instances

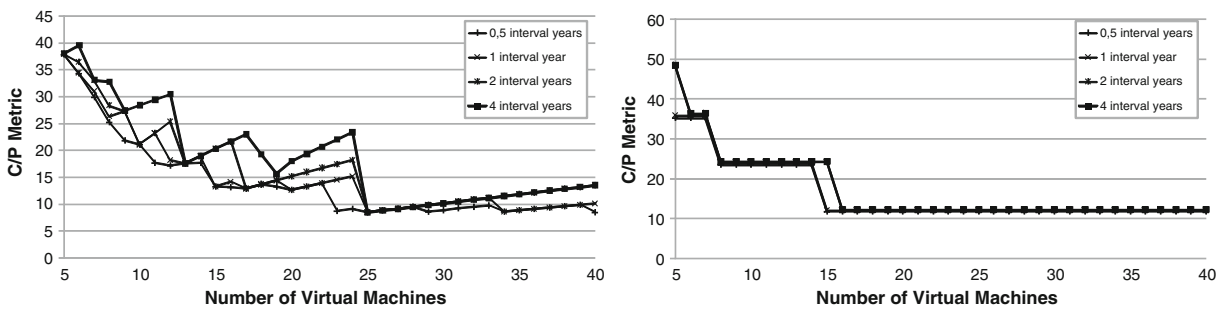
We can conclude that both iCanCloud simulations and the mathematical model show the same tendency in the trade-offs between cost and performance. However, simulations show a more realistic behavior due to more details of the system are modeled. The main advantage of simulations versus the mathematical model is accuracy, and the main drawback is obviously the execution time, because iCanCloud requires to be executed in a computer, which is not required by the mathematical model.

Figure 12 shows a dispersion chart that compares the results of executing the Phobos application in the Amazon EC2 system, against the analogous model using iCanCloud. In those tests both different interval years and a different number of VMs have been used. Results obtained from those tests executed in Amazon EC2 are represented using hollow forms. Otherwise, iCanCloud results are shown using filled forms.

Initially, Phobos application has been launched using 0.5 interval years (represented as squares). This chart shows that both the results executed in the real environment and the results obtained using iCanCloud follows the same tendency. However, those experiments that obtains lower C/P values fits better with the simulation.

Those experiments that use 1 interval year are represented as triangles. In those cases, the C/P value obtained is greater than using a 2 interval year. This is due to the behavior of the Phobos application with different interval years. This asymmetry was already observed during its first porting onto the cloud [33].

In general, in those tests that obtain the lower values of the C/P metric, iCanCloud obtains



(a) iCanCloud (b) Model

Fig. 11 Simulation versus mathematical model of Phobos using high CPU X-Large instances

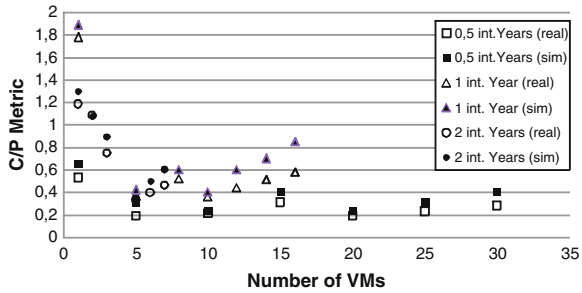


Fig. 12 Simulation versus Amazon EC2 execution of Phobos using small instances

practically the same results. Otherwise, when the C/P value increases, there is a slightly difference between the real system and the simulation. Finally, all results obtained both from the Amazon EC2 and iCanCloud follows a saw-tooth waveform, which can be also appreciated in the previous charts shown in Figs. 7–11.

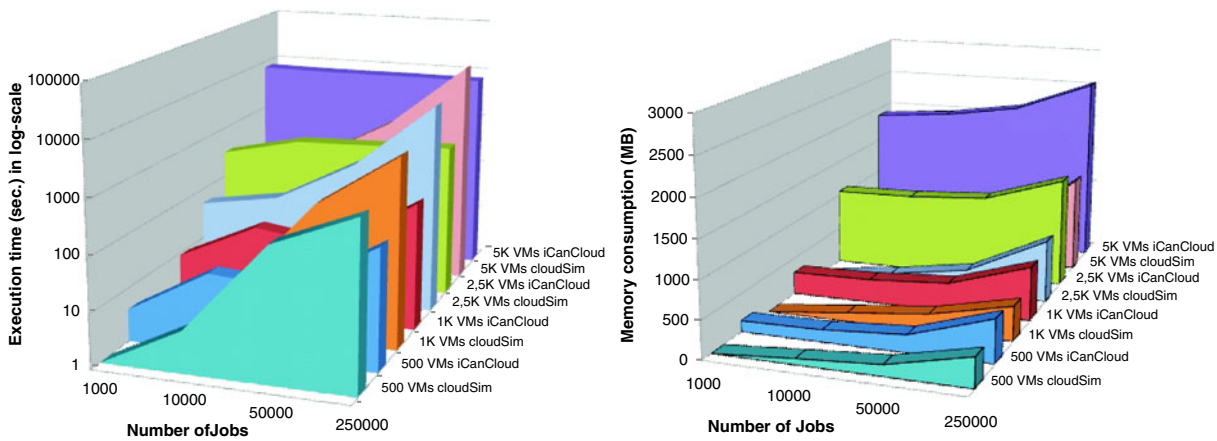
5 Comparison with CloudSim

In order to assess the functionalities of iCanCloud, a set of experiments has been conducted using it and CloudSim with the purpose to compare both simulation tools. We have chosen CloudSim for this comparison because it is a mature simulation tool which has already been used in a number of research works—as Section 2 depicts.

Jobs are modeled in CloudSim by configuring three parameters: (1) input size, (2) processing length, measured in Millions of Instructions (MI), and (3) output size. Input and output sizes refer to the size of input and output files of the job, and is a way to infer the time it takes to execute this job in a cloud resource (whose computing power is measured in Millions of Instructions Per Second—MIPS). Similarly, a new application model has been developed in iCanCloud to execute the same functionality that those jobs do in CloudSim. Thus, the configuration of jobs used in the experiments described in this section is performed equally in both simulation platforms.

Those experiments use jobs whose input size is 5 MB, output size is 30 MB, and processing length is 1,200,000 MI. Also, cloudlets always utilize all the available CPU capacity. These experiments use VMs having 9,500 MIPS, which simulate a standard small instance type provided by Amazon EC2. Those experiments have been executed using an ASUS computer, model u33J Bamboo, which contains a CPU core i3 and 4 GB of RAM memory.

Figure 13 shows the comparison of performance between CloudSim and iCanCloud. Chart 13a shows the execution time of each experiment, where x-axis shows the number of jobs executed in each experiment, y-axis shows the number of VMs and its type, and z-axis shows the time required to execute each experiment (measured in seconds) using a log-scale.



(a) Execution time

(b) Memory consumption

Fig. 13 Performance experiments: iCanCloud versus cloudSim

This chart shows that increasing the number of jobs require more execution time in both simulators, which is obvious. Otherwise, increasing the number of VMs has a different impact in each simulator. In iCanCloud, when more than 2500 VMs are used, the number of jobs has not a considerable impact on the execution time. Note that those experiments executed in iCanCloud have a uniform-like shape. That means that almost CPU power consumed by the simulator is processed for managing VMs. Instead, CloudSim's performance depends directly on both parameters (number of VM and number of jobs). In fact, when the value of one of those parameters increases, execution time increases as well. However, besides some experiments where the number of jobs is less or equal to 50000, iCanCloud is faster than CloudSim. Otherwise, in all tests that use 250K jobs iCanCloud is faster. That means that iCanCloud provides better scalability than CloudSim. The main reason of this is the initialization phase. While iCanCloud requires a considerable amount of time to initialize each component due to the great level of detail, cloudSim performs this phase quickly. However, in large-scale simulations, the time spent in this phase is practically insignificant.

Chart 13b shows the memory consumption of each experiment. In this chart can be appreciated that iCanCloud requires more memory than cloudSim. This is caused because iCanCloud uses a higher level of detail to model each VM instance. As opposed to iCanCloud, which simulates low level details of the cloud system being simulated (as detailed in Section 3), CloudSim does not provide in-depth simulation details. Up to 1000 VMs, the amount of memory required by both simulators is similar. But using more than 1000 Vms, the amount of memory required by iCanCloud grows much faster than CloudSim.

In general, iCanCloud is faster in large scale experiments and provides better scalability, but require more memory than CloudSim.

6 Conclusions and Future Work

In this paper we have introduced the iCanCloud simulator. Its remarkable flexibility, scalability and performance have been proven through validation using an application pertaining to the High Performance Computing domain.

Having set up a complete simulation framework for cloud infrastructures, the present work represents a first step of many. As future work, iCanCloud can be extended in many ways offering interesting research opportunities. For instance, other public cloud infrastructures than Amazon EC2, the one simulated in this paper, could be added. This way, different providers could be compared for an optimal execution of studied applications.

In the present paper we have simulated the execution of an application, but iCanCloud could be extended for simulating complete service levels. Thanks to the simulator's modular design, both Platform as a Service and Software as a Service levels could be defined. Consequently, research on SLAs and Access Control strategies could be conducted using iCanCloud.

As a short term feature, we expect to include parallel simulations to improve performance of very large cloud models. Although iCanCloud has been designed to be parallel, this feature is not completely implemented yet. Moreover, in the mid term we are studying the possibility to include environmental impact in a green computing context.

Finally, private clouds could be easily simulated with an extension of iCanCloud. One of the main directions would be to incorporate a power consumption estimation module (like in CloudSim [7]), enabling the research on GreenIT techniques involving cloud computing.

Software Availability

The iCanCloud simulator is Open Source (GNU General Public License version 3) and it is available at the following website: http://www.iCanCloudSim.org/.

Acknowledgements This research was partially supported by the following projects: Spanish MEC project TESIS (TIN2009-14312-C02-01), and Spanish Ministry of Science and Innovation under the grant TIN2010-16497.

References

1. Bell, W.H., Cameron, D.G., Capozza, L., Millar, A.P., Stockinger, K., Zini, F.: Simulation of dynamic Grid replication strategies in OptorSim. In: Proc. of the 3rd Intl. Workshop on Grid Computing (Grid). Baltimore, USA (2002)
2. Buyya, R., Beloglazov, A., Abawajy, J.: Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. In: Proc of the Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). Las Vegas, USA (2010)
3. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Pract. Ex. **14**, 1175–1220 (2002)
4. Buyya, R., Ranjan, R., Calheiros, R.N.: Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: challenges and opportunities. In: Proc. of the 7th High Performance Computing and Simulation Conference (HPCS) (2009)
5. Calheiros, R.N., Buyya, R., De Rose, C.A.F.: A heuristic for mapping virtual machines and links in emulation testbeds. In: Proc. of the Intl. Conference on Parallel Processing (ICPP). Vienna, Austria (2009)
6. Calheiros, R.N., Buyya, R., De Rose, C.A.F.: Building an automated and self-configurable emulation testbed for Grid applications. Software Pract. Exper. **40**(5), 405–429 (2010)
7. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software Pract. Exper. **41**(1), 23–50 (2011)
8. Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., Gautam, N.: Managing server energy and operational costs in hosting centers. In: Proc. of the Intl Conference on Measurements and Modeling of Computer Systems. Banff, Alberta, Canada (2005)
9. CSIM Development Toolkit for Simulation and Mod-eling: Web page at http://www.mesquite.com/. L a s t a c - cessed 27 July 2011
10. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and Grid computing 360° compared. In: Proc. Grid Computing Environments Workshop (2008)
11. Foster, I.T., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual clusters for Grid communities. In: Proc. of the Sixth Intl. Symposium on Cluster Computing and the Grid (CCGRID). Singapore (2006)
12. Freund, R.F., Gherrity, M., Ambrosius, S.L., Campbell, M., Halderman, M., Hensgen, D.A., Keith, E.G., Kidd, T., Kussow, M., Lima, J.D., Mirabile, F., Moore, L., Rust, B., Siegel, H.J.: Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In: Proc. of the 7th Heterogeneous Computing Workshop (HCW). Orlando, USA (1998)
13. Fujiwara, K., Casanova, H.: Speed and accuracy of network simulation in the simgrid framework. In: Proc. of the 1st Intl. Workshop on Network Simulation Tools (NSTools). Nantes, France (2007)
14. Harri, A., Linkin, V., Pichkadze, K., Schmidt, W., Pellinen, R., Lipatov, A., Vazquez, L., Guerrero, H., Uspensky, M., Polkko, J.: MMPM-Mars MetNet precursor mission. In: European Geosciences Union General Assembly. Vienna, Austria (2008)
15. Kim, K.H., Beloglazov, A., Buyya, R.: Power-aware provisioning of cloud resources for real-time services. In: Proc. of the 7th Intl. Workshop on Middleware for Grids, Clouds and e-Science. Urbana Champaign, Illinois, USA (2009)
16. Kliazovich, D., Bouvry, P., Khan, S.U.: Greencloud: A packet-level simulator of energy-aware cloud computing data centers. J. Supercomput. (2010). doi:10.1007/s11227-010-0504-1
17. Lim, S.H., Sharma, B., Nam, G., Kim, E.K., Das, C.R.: MDCSim: A multi-tier data center simulation, platform. In: Intl. Conference on Cluster Computing and Workshops (CLUSTER). New Orleans, USA (2009)
18. Liu, J., Nicol, D.M.: DaSSF 3.1 User's Manual. Dartmouth College (2001)
19. Liu, X.: Scalable online simulation for modeling Grid dynamics. Ph.D. thesis, Univ. of California at San Diego (2004)
20. Moreno, R., Montero, R., Llorente, I.: Elastic management of cluster-based services in the cloud. In: Proc. of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC), held jointly with the Intl. Conference on Autonomic Computing and Communications. Barcelona, Spain (2009)
21. Núñez, A., Fernández, J., Garcia, J.D., Garcia, F., Carretero, J.: New techniques for simulating high performance MPI applications on large storage networks. J. Supercomput. **51**(1), 40–57 (2010). doi:10.1007/s11227-009-0279-4
22. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: Proc. of the Sixth Intl. Symposium on Cluster Computing and the Grid (CCGRID). Shanghai, China (2009)
23. OPNET modeller: Web page at http://www.opnet.com/. Last accessed 18 Sept 2010
24. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (RAID). In: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, pp. 109–116. Chicago, IL, USA (1988)
25. Romero, P., Barderas, G., Vázquez-Poletti, J.L., Llorente, I.M.: Chronogram to detect Phobos eclipses on Mars with the MetNet precursor lander. Planet. Space Sci. **59**(13), 1542–1550 (2011)
26. Sterling, T.L., Stark, D.: A high-performance computing forecast: partly cloudy. Comput. Sci. Eng. **11**(4), 42–49 (2009)
27. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data Grids: an extension to GridSim. Concurrency and Computation: Pract. Ex. **20**(13), 1591–1609 (2008)

28. The Economist, A Special Report on Corporate IT: Web page at http://www.economist.com/specialReports/showsurvey.cfm?issue=20081025. Accessed 1 Oct 2008
29. The Message Passing Interface (MPI) standard: Web page at http://www.mcs.anl.gov/research/projects/mpi/. Last accessed 18 Sept 2010
30. The Network Animator, Nam: Web page at http://www.isi.edu/nsnam/nam/. Last accessed 27 Jul 2011
31. The Network Simulator, NS-2: Web page at http://www.isi.edu/nsnam/ns/. Last accessed 18 Sept 2010
32. Varga, A.: The OMNeT++ discrete event simulation system,. In: Proc. of the European Simulation Multiconference (ESM). Prague, Czech Republic (2001)
33. Vázquez-Poletti, J.L., Barderas, G., Llorente, I.M., Romero, P.: A Model for Efficient onboard actualization of an instrumental cyclogram for the Mars MetNet mission on a public cloud infrastructure. In: Proc. of PARA: State of the Art in Scientific and Parallel Computing (Lecture Notes in Computer Science), pp. 33–42. Reykjavík, Iceland (2012)
34. Vouk, M.A.: Cloud computing: Issues, research and implementations. In: Proc. of the 30th Intl. Conference on Information Technology Interfaces (ITI). Dubrovnic, Croatia (2008)
35. Wang, Z., Zhu, X., McCarthy, C., Ranganathan, P., Talwar, V.: Feedback control algorithms for power management of servers. In: Proc. of the Third International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBid). Annapolis, USA (2008)
36. Wickremasinghe, B., Calheiros, R.N., Buyya, R.: Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: Proc. of the 24th Intl. Conference on Advanced Information Networking and Applications (AINA). Perth, Australia (2010)